Project Presentation: Algorithmic Structuring and Compression of Proofs (ASCOP)

Stefan Hetzl

Project Presentation: Algorithmic Structuring and Compression of Proofs (ASCOP)

Stefan Hetzl

Institute of Discrete Mathematics and Geometry
Vienna University of Technology
Wiedner Hauptstraße 8-10, A-1040 Vienna, Austria
hetzl@logic.at

Abstract. Computer-generated proofs are typically analytic, i.e. they essentially consist only of formulas which are present in the theorem that is shown. In contrast, mathematical proofs written by humans almost never are: they are highly structured due to the use of lemmas.
The ASCOP-project aims at developing algorithms and software which structure and abbreviate analytic proofs by computing useful lemmas. These algorithms will be based on recent groundbreaking results establishing a new connection between proof theory and formal language theory. This connection allows the application of efficient algorithms based on formal grammars to structure and compress proofs.

1 Introduction

Proofs are the most important carriers of mathematical knowledge. Logic has endowed us with formal languages for proofs which make them amenable to algorithmic treatment. From the early days of automated deduction to the current state of the art in automated and interactive theorem proving we have witnessed a huge increase in the ability of computers to search for, formalise and work with proofs. Due to the continuing formalisation of computer science (e.g. in areas such as hardware and software verification) the importance of formal proofs will grow further.

Formal proofs which are generated automatically are usually difficult or even impossible to understand for a human reader. This is due to several reasons: one is a potentially extreme length as in the well-known cases of the four colour theorem or the Kepler conjecture. But one need not go that far to make this point, a quick glance at the output of most of the current automated theorem provers may very well suffice to demonstrate this difficulty. In such cases, where mere size is not problematic, one faces logical issues such as the use of deduction formats more suited for finding than for representing proofs as well as engineering issues such as user interfaces.

Below all these aspects however is lurking a reason of a much more fundamental nature: computer-generated proofs are *analytic*, i.e. they essentially only contain such formulas which are already present in the theorem that is shown. In contrast, human-generated mathematical proofs almost never are; in

a well-structured proof the final result is usually derived from lemmas. Indeed, the computer-generated part of the proof of the four colour theorem as well as that of the the Kepler conjecture is – from the logical point of view – essentially the verification of a huge case distinction by calculations, a typical form of an analytic proof. With increasing automation in many areas, the share of such proofs can be expected to grow, another recent example being the solution to the Sudoku Minimum Number of Clues problem [15].

Such inscrutable analytic proofs *do* carry mathematical knowledge, after all they show that the theorem is true. However they carry this knowledge only in an implicit form which renders it inaccessible (to a human reader). The aim of the ASCOP-project is to develop methods and software which makes this knowledge accessible by making it explicit in the form of new lemmas.

2 Theoretical Foundations

Since the very beginning of structural proof theory, marked by the seminal work [6], it is well understood that arbitrary proofs can be transformed into analytic proofs and how to do it. This process: cut-elimination, has been the central axis of the development of proof theory in the 20th century (the inference rule *cut* formalises the use of a lemma in a proof). The approach of the ASCOP-project is to develop algorithms which *reverse* this process: starting from an analytic proof (e.g. one that has been generated algorithmically) the task is to transform it into a shorter and more structured proof of the same theorem by the introduction of cuts which – on the mathematical level – represent lemmas. An algorithmic reversal of cut-elimination is rendered possible by recent groundbreaking results (like [10, 8] and [12], see also [9]) that establish a new connection between proof theory and formal language theory.

For explaining this connection, let us first consider one of the most most fundamental results about first-order logic: Herbrand's theorem [7]. In its simplest form it states that $\exists x\, A$, for a quantifier-free formula A, is valid iff there are terms $t_1, \ldots, t_n$ s.t. $\bigvee_{i=1}^{n} A[x\backslash t_i]$ is a propositional tautology. A disjunction of instances of a formula which is a tautology is therefore also often called *Herbrand-disjunction*. This theorem can be greatly generalised (see e.g. [16]), for expository purposes we stick to formulas of the form $\exists x\, A$ here. A Herbrand-disjunction corresponds to a cut-free proof in the sense that $\exists x\, A$ has a cut-free proof with n quantifier inferences iff it has a Herbrand-disjunction with n disjuncts. We write $\mathrm{H}(\pi)$ for the set of disjuncts of the Herbrand-disjunction induced by the proof π.

It is well-known that cut-elimination may increase the length of proofs considerably, e.g. in first-order logic the growth rate is 2_n where $2_0 = 1$ and $2_{i+1} = 2^{2_i}$. Now, if a large Herbrand-disjunction arose from eliminating the cuts of a small proof, then this Herbrand-disjunction must necessarily contain a certain amount of regularity / redundancy because it has a short description: the original proof with cuts. While this observation is obvious, the question what that redundancy is and how it can be characterised and detected is much less so.

This question has recently been answered in [10, 8] for the case of proofs with Σ_1-cuts, i.e. proofs whose cut formulas have a prenex normal form $\exists x\, B$ where B is quantifier-free (note that a cut on a formula $\forall x\, B$ can easily be transformed to a Σ_1-cut by adding a negation and switching the left and right subproofs). In [10, 8] it is shown that a Herbrand-disjunction that arose from a proof π with Σ_1-cuts can be written as the language of a **totally rigid acyclic tree grammar** that has the size of π. Rigid tree languages have been introduced in [13] with applications in verification in mind (e.g. of cryptographic protocols as in [14]). A rigid tree grammar differs from a regular tree grammar (see e.g. [5]) in that it allows certain equality constraints. Totally rigid acyclic tree grammars are a subclass of them, see [10, 8] for details. Such results that describe the structure of Herbrand-disjunctions depending on the class of proofs with cut from which they originate will be called *structure theorems* in the sequel.

$$\begin{array}{ccc} \text{proof } \pi \text{ with cuts} & \longrightarrow^{\text{cut-elimination}} & \text{cut-free proof } \pi^* \\ \downarrow & & \downarrow \\ \text{grammar } \mathrm{G}(\pi) & \longrightarrow^{\text{defines}} & \text{language } \mathrm{L}(\mathrm{G}(\pi)) = \mathrm{H}(\pi^*) \end{array}$$

Fig. 1. Proofs and Tree Grammars

What has thus been obtained is a correspondence as depicted in Figure 1: on the level of Herbrand-disjunctions, cut-elimination is nothing but the computation of the language of a totally rigid acyclic tree grammar. Consequently this structure theorem tells us what we have to look for in a Herbrand-disjunction if we want to abbreviate it using Σ_1-cuts: we have to **write it as the language of a totally rigid acyclic tree grammar!**

These results suggest the following systematic approach to the design of proof compression algorithms: a first theoretical step consists in proving a structure theorem for a particular class of proofs with cut. The proof compression algorithm is then designed to start from a Herbrand-disjunction H and proceed in two phases:

- First, a grammar that represents H is computed. This is a pure term problem consisting of finding a minimal (w.r.t. the number of productions) grammar for a given finite language (i.e. a trivial grammar). This problem is closely related to automata minimisation, one of the most standard problems in formal language theory.
- Secondly, cut formulas that realise this grammar in the form of a proof with cuts are computed. In the case of a single Σ_1-cut there is always a canonical solution which is computable in linear time [11]. This property carries over to an arbitrary number of Σ_1-cuts and – a priori – there is no reason to assume a different behaviour in the general case.

Furthermore, one obtains a completeness result of the following form: if there is a proof with cuts that leads to H via cut-elimination, the above algorithm finds

it (note the contrast to the undecidability of k/l-compressibility [3]). Therefore one also obtains a maximal compression: the algorithm finds the proof with the smallest grammar that leads to a given cut-free proof.

A first proof-of-concept algorithm realising this approach for the class of proofs having a single Σ_1-cut is presented in [11].

3 Aims of the ASCOP-Project

The purpose of the ASCOP-project is to fully exploit the potential of this approach to structuring and compression of proofs. On the theoretical side, our main aim is to extend the classes of lemmas that can be computed beyond those in [11]. Preliminary investigations show that this extension is rather straightforward as long as the lemmas do not contain quantifier alternations. To treat those, an extension of the theoretical results of [10, 8] is necessary first. As a bridge to practical applications it will also be useful to generalise these algorithms to work modulo simple theories such as equality for uninterpreted function symbols or linear arithmetic.

We will implement these proof compression algorithms based on the GAPT-project [1]. GAPT (Generic Architecture for Proofs) is a general framework for proof-theoretic algorithms implemented in Scala. Its primary application is to serve as a basis for the CERES-system [4], a system for the analysis of formalised mathematical proofs based on resolution provers. As an appropriate frame for these algorithms we envisage an implementation that allows to use the output of a resolution theorem prover as input and to compute a sequent calculus proof with cuts of the same theorem. Frequently, the user will primarily be interested in the computed lemmas, viewing the complete proof being only an option for a more detailed analysis. The existing graphical user interface of GAPT provides an adequate basis for a sufficiently flexible user interaction. As a large-scale test of our algorithms we plan to apply them as post-processing step to the output of standard resolution provers on the TPTP library [17], as in [18].

The ASCOP-project envisages a varied range of applications. In the short term we expect the system to be useful for improving the readability of the output of automated theorem provers. We furthermore expect these simplification and compression capabilities to be useful for the integration of automated provers in proof assistants (such as sledgehammer in Isabelle [2]) as they allow to break up automatically generated proofs into smaller pieces (thus facilitating their replay by Isabelle's trusted resolution prover metis). In the long term we hope that these methods have the potential to compute mathematically meaningful information from large and inscrutable analytic proofs such as that of the four colour theorem, the Kepler conjecture, the Sudoku clues proof and other similar proofs to be expected to surface in the future.

The reader interested in following the progress of the ASCOP-project is invited to consult its website at http://www.logic.at/people/hetzl/ascop/.

References

1. Generic Architecture for Proofs (GAPT). http://code.google.com/p/gapt/
2. Sledgehammer. www.cl.cam.ac.uk/research/hvg/Isabelle/sledgehammer.html
3. Baaz, M., Zach, R.: Algorithmic Structuring of Cut-free Proofs. In: Computer Science Logic (CSL) 1992. Lecture Notes in Computer Science, vol. 702, pp. 29–42. Springer (1993)
4. Dunchev, T., Leitsch, A., Libal, T., Weller, D., Woltzenlogel Paleo, B.: System Description: The Proof Transformation System CERES. In: Giesl, J., Hähnle, R. (eds.) 5th International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Computer Science, vol. 6173, pp. 427–433. Springer (2010)
5. Gécseg, F., Steinby, M.: Tree Languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages: Volume 3: Beyond Words, pp. 1–68. Springer (1997)
6. Gentzen, G.: Untersuchungen über das logische Schließen I. Mathematische Zeitschrift 39(2), 176–210 (1934)
7. Herbrand, J.: Recherches sur la théorie de la démonstration. Ph.D. thesis, Université de Paris (1930)
8. Hetzl, S.: Proofs as Tree Languages, submitted, preprint available at http://hal.archives-ouvertes.fr/hal-00613713/
9. Hetzl, S.: On the form of witness terms. Archive for Mathematical Logic 49(5), 529–554 (2010)
10. Hetzl, S.: Applying Tree Languages in Proof Theory. In: Dediu, A.H., Martín-Vide, C. (eds.) Language and Automata Theory and Applications (LATA) 2012. Lecture Notes in Computer Science, vol. 7183, pp. 301–312. Springer (2012)
11. Hetzl, S., Leitsch, A., Weller, D.: Towards Algorithmic Cut-Introduction. In: Logic for Programming, Artificial Intelligence and Reasoning (LPAR-18). Lecture Notes in Computer Science, vol. 7180, pp. 228–242. Springer (2012)
12. Hetzl, S., Straßburger, L.: Herbrand-Confluence for Cut-Elimination in Classical First-Order Logic, submitted
13. Jacquemard, F., Klay, F., Vacher, C.: Rigid tree automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) Third International Conference on Language and Automata Theory and Applications (LATA) 2009. Lecture Notes in Computer Science, vol. 5457, pp. 446–457. Springer (2009)
14. Jacquemard, F., Klay, F., Vacher, C.: Rigid tree automata and applications. Information and Computation 209, 486–512 (2011)
15. McGuire, G., Tugemann, B., Civario, G.: There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem, http://arxiv.org/abs/1201.0749
16. Miller, D.: A Compact Representation of Proofs. Studia Logica 46(4), 347–370 (1987)
17. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning 43(4), 337–362 (2009)
18. Vyskočil, J., Stanovský, D., Urban, J.: Automated Proof Compression by Invention of New Definitions. In: Clark, E.M., Voronkov, A. (eds.) Logic for Programming, Artifical Intelligence and Reasoning (LPAR-16). Lecture Notes in Computer Science, vol. 6355, pp. 447–462. Springer (2010)